AEC RESEARCH AND
DEVELOPMENT REPORT

NYO-6478

NEW YORK UNIVERSITY
INSTITUTE OF MATHEMATICAL SCIENCES
LIBRARY
25 Waverly Place, New York 3, N. Y.

N-10647B

NEW YORK UNIVERSITY

institute of mathematical sciences

AEC Computing Facility

NEW YORK UNIVERSITY
INSTITUTE OF MATHEMATICAL SCIENCES
25 Waverly Place, New York 3, N. Y.

New York, New York

NYO-6478

NEW YORK UNIVERSITY COMPILER SYSTEM

Roy Goldfinger

February 10, 1954

New York University
AEC Computing Facility

AT(30-1)-1480

Issued
March 1, 1954

opp

INDEX

Part I

THE COMPILER

Section 1.	The Compiler	Page 1
Section 2.	Subroutines	3
Section 3.	The Directory	8
Section 4.	Operating Instructions	13
Section 5.	Compiler Logic	15
Section 6.	Modification I	18

Part II

COMPILER SYSTEM

The Librarian	20
Subroutine Corrector	22
Mike I	23
Floating Decimal - 1	24

1 March 1954

rev. 1 August 1954

FOREWORD

Why automatic programming? One motivation perhaps may be a scarcity of competent programmers for the solution of problems, which, Dr. Wilkes[1] suggests, may be due in turn to the fact that once programmers achieve competence they become more interested in developing automatic programming techniques than in solving problems. In any event, the desire, and perhaps the need, to increase the degree of automatization in programming is very much with us and should be met. It is also in our interest to devise means by which the programming and coding for our machine (a Univac) can be reduced in its complexity; and need for attention to detail, with a view toward making the machine available to persons whose primary interest is not in the machine but in the problem.

The system presented here is an approach, an attempt to streamline programming and mechanize some aspects of coding. Much of the inspiration and a good part of the technique derives from an appreciation of the aims and achievements of Dr. Grace M. Hopper and her staff at Remington Rand, Inc.

Roy Goldfinger

---

(1) Dr. M. V. Wilkes, at the Cambridge meeting of the Association for Computing Machinery, September 9, 10, 11, 1953.

opp

THE COMPILER

1,1 This manual describes a compiler which was developed at the AEC Computing Facility, New York University, for use with the Univac. The compiler was made available in November, 1953, and has since been used successfully in the preparation of several programs of varying size. Associated with the compiler are some routines of a "service routine" nature which are designed specifically to process the kind of coding with which the compiler deals. At the time of this writing work is in progress on other subordinate routines, and on library routines of general utility. This manual then is an "open-ended" description of a system, and it is expected that supplementary material will be made available in the near future.

1.2 A compiler is a program which takes as input a number of isolated chunks of coding (called "subroutines") and assembles them, suitably modified, into a "running program" which, when returned to the computer, is capable of solving a problem. Once a running program is achieved as an end product of the compilation, the compiler is no longer related to the running of the problem except as the character of the resultant program reflects the compiler that produced it.

The nature and extent of the effect of the compiler upon the resultant program is a matter of some concern. The end result should not be limited by its "inherited characteristics", or put another way, the final program produced by the compiler should not be inferior to another program, for solving the same problem, produced without the aid of a compiler.

In the system presented here particular attention has been given to the matter of avoiding undesirable "inherited characteristics." The aim has been to insure that the running program produced by the compiler be as good a program as could be written independently of a compiler. One way in which this was attempted was to grant the programmer the greatest possible amount of control

opp

over the behavior of the compiler, even to the extent of requiring that the programmer assume clerical tasks which the machine could be made to accomplish easily. It is sometimes more profitable to require less of the machine. Where the machine might proceed in some inflexible way to assign memory locations in the running program, for example, the programmer, if given control, may arrange the memory as he wishes.

What has resulted from these considerations is a compiler with the following characteristics:

A. The compiler produces coding which line for line (with rare exceptions) is identical with straight Univac coding.

B. It makes the entire memory available to the running program.

C. It places no restriction on the extent of cross references between subroutines.

D. It permits both the program tape and the memory to be organized in whatever manner is best suited to the problem.

1.3 The general procedure which the programmer follows in using the compiler is as follows:

A. The programmer plans to utilize subroutines already available in the general library.

B. He writes the coding which is unique to his problem as one or more subroutines according to the rules in Section 2 of this manual.

C. He prepares a "directory", as described in Section 3, which specifies how the program tape and the memory shall be organized.

D. He utilizes the Univac under control of the compiler to write the running program.

The result of step D is a tape which is ready to be run, or at least tested, on the Univac.

1.4 The following sections of this manual discuss the use of the compiler in detail.

Section 2

SUBROUTINES

2.1 Introduction and definition of terms. The basic function of the compiler is to assemble subroutines. The term "subroutine" does not admit of precise definition. It can be given a functional meaning which ascribes to a subroutine the potentiality for performing a particular task essential to the completion of a program. While this is the usual definition it does not entirely suit us when we consider subroutines in the light of compilers. It is far more convenient here to consider subroutines as the ingredients which the compiler compiles into a program, leaving it to the programmer to determine for himself what he shall call a subroutine. For example, if it is a particular programmer's practice to consolidate his constants in a separate section of the memory he will find it convenient to call his set of constants a subroutine and compile it in the same way that he compiles the other basic components of his program.

The unit element on which the compiler operates is the individual line of coding within the subroutine. A line of coding consists of the customary left hand and right hand instruction to the Univac expressed in the C-10 code, together with an additional character or characters imbedded in certain of the lines indicating how the addresses on that line shall be modified to make the subroutine capable of functioning correctly anywhere in the memory. Each subroutine is originally coded in relative form in which the first line of the routine is assigned the memory location number 000 and succeeding lines numbered consecutively thereafter.

2.2 The label line. Line 000 of the subroutine is reserved for the label line which carries the label of the routine, its number of compilable lines, and a sentinel which identifies it as the first line of a subroutine. A typical label line is:

PQO 028 000 00/

in which PQ is the two-Univac-character label assigned the routine, 028 is the number of lines in the routine which will be transcribed onto the program tape, and the terminal ignore is the sentinel which identifies this word as the label line of a subroutine in the library. NOTE: Where a subroutine is longer than one block it is not permissible to allow an ignore to appear as the 12th digit of the first word of succeeding blocks of the routine.

2.3 Internal, external, and absolute references. The address portion of an order on a line in a subroutine can have one of several meanings. It can be irrelevant (as in the K instruction), or it can refer to a part of the memory which is always fixed, such as a line of the floating decimal routine (which has been frozen in 880 to 999) or perhaps 000 in setting the overflow constant. Such an address is called an absolute reference, and is not meant to be modified by the compiler.

The kind of address with which the compiler is more concerned is called a relative address, one which is meant to be modified by the compiler. Such an address always refers to a line of a subroutine relative to its starting line. It can refer to a line in the subroutine in which it is located, or it can refer to some line in an entirely different subroutine. In the former case it is called an internal reference, and in the latter case an external reference. The 9th digit position in a line of coding which contains a relative address is utilized to indicate to the compiler how that line shall be modified.

For internal references:

L in the 9th digit position indicates that the left hand address is relative.

R in the 9th digit position indicates that the right hand address is relative.

W in the 9th digit position indicates that the whole line is relative.

For external references:

X in the 9th digit position indicates that the whole line refers relatively to another subroutine.

The 2nd and 3rd digit positions carry the label of the subroutine to which reference is made.

A function of the compiler is to modify each relative address by addition of a value which will change it into an absolute address in the running program.

2.4 Sentinel line. The ending sentinel for a subroutine is a word of printer breakpoints ~~BBBBBB BBBBBB~~. It appears on the line of the subroutine which corresponds to the line number in the 4th, 5th, and 6th digits of the label line.

2.5 Example. In the following example it is assumed that routine A2 has transferred control to routine SC (Servo change) line 001, and that SC will return control to A2 after modifying two of its lines.

```
000   SC0 011   000 00~~1~~
001   50  009   10W 010
002   B   010   FOW 007
003   EA2 004   HOX 004
004   ;6  000   FOR 008
005   EA2 005   COX 005
006   0A2 000   UOX 004
007   101 111   101 111
008   101 111   111 111
009   WHA T△S   ERV 0△△
010      -         -
011   ~~BBB~~ ~~BBB~~   ~~BBB~~ ~~BBB~~
          FILL
```

opp

2.5 continued.

Under the assumption that routine SC is compiled so as to start in memory location 135 and A2 in 014, that part of the program tape into which SC is compiled will look as follows:

135	000	SC0	000	011
136	50	144	10	145
137	B	145	F	142
138	E	018	H	018
139	;6	000	F	143
140	E	019	C	019
141	000	014	U	018
142	101	111	101	111
143	101	111	111	111
144	WHA	T⊿S	ERV	0⊿⊿
145	-		-	

In this example note particularly the transformation of the label line, and the fact that there is a line of coding on the program tape corresponding to each line in the uncompiled subroutine not including the line of printer breakpoints. Line 146 in the compiled program will be the label line (transformed) of the next subroutine which is compiled.

2.6 <u>The library tape</u>. Subroutines are stored on a tape which is called the library tape. There is an initial sentinel block whose first word is -ZZ ZZZ ZZZ ZZ/, and a final sentinel block whose first word is ZZZ ZZZ ZZZ ZZ/. The contents of the remainder of these two blocks is of no significance. Between the two terminal blocks the subroutines are stored in order of increasing Univac magnitude with respect to label lines.

The compiler can take its input of subroutines from either of two library tapes. To standardize procedure it is recommended that the tape which consists of general library routines be mounted on servo 6, while the tape which contains the subroutines which are unique to the problem being compiled be on servo 3.

opp

2.7 Special considerations. It should be apparent that two different subroutines cannot occupy parts of the same block on the library tape, and that the label line of a subroutine is always in word 00 of the first block of the routine. It follows that subsequent blocks of a subroutine should not have an ignore in the 12th digit position of the first word of the block.

It is not possible to refer relatively to two different subroutines in the same line of coding. However reference can be made to an external routine and an absolute address by coding the absolute address in fourth zone alphabetic equivalents of first zone numerics. (The Univac system at NYU has been modified to permit the use of a fourth zone equivalent for zero.) The compiler will convert such alphabetic addresses to their numerical equivalents whenever they occur as a part of an external reference.

THE DIRECTORY

3.1 Introduction. The directory provides all of the control information required for a compilation. It tells the compiler which subroutine to seek on the library tape, on which library tape to look, and how to modify the routine it finds.

The order in which subroutines are placed on the program tape is determined by the order in which they are listed in the directory. Should it be necessary to begin a subroutine on a multiple-of-10 line (to preserve latency time, perhaps), or at a block beginning, or just to leave expansion room in the compiled program, there are directory items which cause partial or full fills (with words of skips) of the program tape.

There are several other directory items recognized by the compiler which together with the ones already mentioned provide for a highly flexible system for organizing and controlling the contents of the program tape and therefore of the running memory. Section 3.3 lists and analyzes the various directory items.

3.2 General considerations. The directory may be either prepared beforehand on tape or typed directly into the memory at the time of compilation. In either case it is limited in length to three blocks. The last item of the directory is always a word of Z's.

In making up the directory the programmer determines for himself how he wishes to utilize the running memory of the problem. The order in which directory items call for subroutines, together with the tape read orders on the program tape determine the running memory. Relative references within the subroutines will be adjusted by the compiler according to the starting line numbers listed in the directory.

No consistency check is applied by the compiler to the directory as to the organization of the memory. At the discretion of the programmer routines may be overlapped; several memory loads may be organized; all the tricks of programming may be employed. For this reason the programmer must exercise care in assigning the starting line numbers in the directory. Usually he must insure that his read-in orders place subroutines into memory locations for which addresses have been adjusted by the assignment of the starting line numbers. The compiler will produce a program tape whenever (1) the directory is not lacking in referents for each external reference encountered, and (2) the tape search yields the desired subroutines.

3.3 Analysis of directory items. Typical items in the directory together with their effect upon the compiler are listed below:

PQ0 000 078 184 (D1)

a) PQ is the label of the subroutine to be compiled.
b) Routine PQ is stored in the library on servo 6.
c) There are 78 lines in routine PQ (label line: PQ0 078 000 00~~1~~) which will be transcribed onto the program tape on servo 5.
d) Each address in the program which refers relatively to subroutine PQ will be modified by an increment of 184.

ABZ 000 049 210 (D2)

a) Similar to item type (D1) except that subroutine AB is stored in the library on servo 3.

C1Z CCC 050 300 (D3)

a) Routine C1 will be transcribed from servo 3 to servo 5 without attention by the compiler to characters in the ninth digit position. Lines of coding in routine C1 will not be modified.
b) External references to C1 in other parts of the program will be modified by an increment of 300.

3.3 continued.

S10 SSS 000 600 (D4)

a) Working storage area S1 begins in memory location 600. External references to area S1 are modified by an increment of 600.
b) Since S1 does not exist as a subroutine, library tape will not be searched.

000 FFF 000 000 (D5)

a) Fill the remainder of the block currently being written on the program tape with skips.
b) Repeated (D5) items cause additional blocks of skips to be written.

000 PPP 000 000 (D6)

a) Fill the remainder of the current 10-word sub-block of the program tape with skips so that the next subroutine compiled starts on a multiple-of-10 line.
b) Repeated (D6) items cause additional sub-blocks of skips to be written.

000 FDR 000 000 (D7)

a) Write the two-block floating decimal arithmetic (FD-1) on the program tape. FD-1 is taken from the compiler instruction tape.
b) The compiler will insure that FD-1 starts a new block on the program tape by filling the previous block with skips, if necessary.
c) FD-1 is always used in memory locations 880 to 999.

ZZZ ZZZ ZZZ ZZZ (D8)

a) Terminal item of the directory.
b) Fills the remainder of the last block with skips.

3.4 <u>Example</u>. Program A.

Directory:		Comments:
00	A1Z 000 006 000	Read-in orders (see below) - Usually the last routine written.
01	A2Z 000 178 006	

3.4 continued.

Directory: (continued)		Comments: (continued)
02	000 PPP 000 000	Partial fill because the routine which follows is coded for minimum time.
03	EX0 000 104 190	Routines EX and +1 are library routines.
04	+10 000 038 294	
05	000 FFF 000 000	
06	A3Z CCC 060 350	Lumped constants for routine A2.
07	000 FDR 000 000	
08	S10 SSS 000 600	Working storage - necessary item in the directory only if S1 is referred to relatively.
09	ZZZ ZZZ ZZZ ZZZ	

Subroutine A1:

000	A10 006 000 00~~1~~	
001	11 000 31 060	Read-in orders for Program A, reading in the program in the order in which the component parts were put on the program tape as given in the directory.
002	31 120 31 180	
003	31 240 31 300	
004	31 350 31 880	
005	30 940 61 000	
006	~~000~~ ~~000~~ ~~000~~ ~~000~~	
	FILL	

3.5 <u>Comments</u>. In the example given in the previous section it might be noted that several economies could be effected in the directory. Item 5, for example, is not necessary since a fill of the remainder of the 300 block on the tape could have been accomplished by calling for the floating decimal routine before calling in the constants (subroutine A3) rather than after. The tape-read

opp

3.5 continued.

orders of subroutine A1 would then be changed to read blocks into 880 and 940 before reading a block into 350 of the memory.

Line 8 of the directory in the example can also be omitted in the following way: The purpose of line 8 is to carry the information that S1 begins in memory location 600. The SSS serves only to inhibit tape search for a routine called S1. However, directory items of the form PPP, FFF, and FDR can be made to carry the same information as line 8 while still serving their original function. Thus, line 2 of the sample directory could read S10 PPP 000 600, and line 8 omitted.

There are occasions when it might be convenient to leave blank space in a directory on tape for future filling in. The item 000 SSS 000 000 will produce no effect on the program tape provided that there are no external references to a routine 00.

The programmer will conceive of other devices by which the directory can be made to provide added flexibility in the way in which a program can be organized. This is particularly true as the program increases in length, running to more than one memory load. Within a few lines of directory items the programmer can specify that certain subroutines be repeated on the program tape (in different memory locations in each memory load, perhaps) while others are frozen in the memory. Lengthy program tapes can be written with a considerable economy of control information. Experience together with an understanding of the logic of the compiler will supply the solution to many problems in program organization.

OPERATING INSTRUCTIONS

4.1 Servos

1. Compiler System.
2. Directory (unless typed-in manually).
3. Library of subroutines (coding unique to the problem).
5. Blank (to receive compiled program).
6. Library of subroutines (general library routines).

4.2 Breakpoints

4. Normal, no transfer reads in directory from servo 2.

   Force transfer to type in directory from supervisory control.

5. Force transfer to change a directory item.

4.3 S.C. typewriter

Normal.

4.4 Procedure

Initial read the compiler and use breakpoints 4 and 5 as indicated.

Clear C at any time to rewind all tapes and initial read locator.

4.5 Typeouts

Normally the only typeouts of significance are the directory items. Each item is typed out just before breakpoint 5

However, the following error typeouts can occur. [-] denotes "contents of".

"REPLACE SKP" occurs if a line of the directory read in from servo 2 contains 12 zeros. Compiler calls for a type-in to memory location 820 plus the number of the directory item.

4.5 Typeouts (continued) "NOT IN LIB- [Directory item]" occurs if the subroutine called for could not be found on the tape specified. All tapes rewind and compiler stops. Clear C and recompile, forcing transfer on breakpoint 5 to correct the faulty item.

"NOT IN DIR- [Directory item] [External reference line]" occurs if the referent of some external reference cannot be found in the directory. All tapes rewind and compiler stops.

"NO BKPTS- [Directory item]" occurs if the compiler cannot find the line-of-breakpoints sentinel at the end of the subroutine. All tapes rewind and the compiler stops.

4.6 Note. Whenever the compiler is run with the supervisory control output switch in the "skip" position, it is desirable to place the breakpoint switch in "breakpoint" position. A comma breakpoint instruction occurs in the coding prior to each of the error typeouts.

SUPPLEMENTARY

5.1 An understanding of the construction and logic of the compiler is certain to lead to more advantageous use of the subroutine method of programming as well as to the continuing discovery of new techniques in the use of the compiler and related routines. This part of the Compiler Manual will outline the construction of the compiler. To pursue the subject further it would be desirable to obtain and study the flow diagrams and coding.

5.2 The compiler viewed overall embraces subroutines A0 through A2, AA through AG, and CC. Briefly stated, each subroutine fulfills the following functions:

Subroutine A0 reads in the instruction tape, sets the clear C option, and initializes counters to eliminate the necessity for a cleared memory.

Subroutine A1 obtains and stores the directory in the memory by type-in or tape-read according to the option selected at breakpoint 4. The tape read-in of a word of zeros is rejected and a replacement called for. A full word of Z's terminates the loop in either option.

Subroutine A2 is the main control or switching center of compiler activity.

Subroutine AA searches a library tape for a subroutine called for in the directory. A "Z" in the directory item calls for a read of Servo 3; otherwise Servo 6 is searched. The direction of read is determined by examining the next label line on the tape for magnitude with respect to the label line desired. Tape is then read in the selected direction until either the routine is found or an end sentinel block on the tape is picked up. In the latter case the error type-out "NOT IN LIB-" occurs, all tapes rewind and the compiler shuts down. Too many possibilities for error exist to suggest a

unique course of corrective action for this error.

Subroutine AB searches the directory from first line to Z's for a referent for each external reference encountered in the coding being compiled. Only the first two digits of each directory item are checked against external reference labels. The last three digits in the item on which agreement is reached are taken to be the starting line to be added to both halves of the external reference line. If the referent is not found the type-out "NOT IN DIR-" occurs; compiler shuts down.

Subroutine AC picks up lines of the subroutine undergoing processing. When the end of the input block is reached without the terminal sentinel of printer breakpoints having been sensed, subroutine AC checks against the line count number in the label line to determine whether there should be an additional block after the present one. If not, the typeout "NO BKPTS-" occurs. Note that this error detector is quite easy-going, accurate only to within a block.

Subroutine AD places a single word in the output block and writes on tape 5 when the block is filled. It also records the count of blocks accomplished in mem. loc. 711. This number is not typed out by the compiler, but is available if desired.

Subroutine AE is the sensory center of the compiler. It examines each directory item in turn to determine what mode of operation is being requested. The sensitive 4th, 5th, and 6th digits of the item are tested first, and if they are greater than zero it is determined that something other than a straightforward tape-search, process-subroutine sequence is called for. Equality tests of the three digits against CCC, FFF, PPP, FDR, SSS, and ZZZ occur, one of which, it passed, produces the effect described in Section 3 of this manual. Note that the test is against three digits only. If none of these tests is passed control is transferred to Subroutine AG.

Subroutine AF performs fill operations.

Subroutine AG is Modification I, discussed in Section 6. It employs AA to locate the routine desired, reads it forward to its last block, searches for the terminal sentinel line and transfers the keys out of the input block to a safer place in the memory. In using the keys to determine where additions in the subroutine being processed are to take place, it proceeds backwards through the keys, writing the processed blocks in reverse order on Servo 4. Subroutine AG then sets Subroutine AC to read Servo 4 backward during compilation instead of a library tape forward.

Subroutine CC contains lumped constants.

5.3 All of the task routines AA through AG are entered with an R - U ins instruction from the main control section. Only AC has a variable exit depending on whether the last line of the subroutine has been reached. All subroutines use the common store of constants in CC, and a common working store in memory locations 700 through 715.

MODIFICATION I

6.1 The function of Mod I in the compiler is to permit the transfer, at the time of compilation, of half or whole lines of information, 'parameters', from the directory into preselected lines of the subroutine. The modification was motivated by the need to have generalized routines available in the tape library which could be altered by the compiler under directory control to 'particularize' them for the problem being compiled.

6.2 The subroutine is generalized by having whole or half lines of coding in which undetermined digit values are stored as zeros. For example, a tape rewind order for an undetermined servo becomes 60,000. Such a half or whole line is termed a 'blank' in the subroutine. The blanks are filled by the compiler by adding half or whole words from the directory.

The 'keys' to the blanks are stored in the lines following the printer breakpoint sentinel at the end of the subroutine. Each key is a half word; there is a key for each addition which is to occur. For example, if line 036 of the subroutine is 60 000 K 000, the corresponding key might be 036L01, where L states that the left half of 036 is a blank (R for right, W for whole line) and 01 specifies in which directory line the additive to 036 is to be found - see below.

The keys are terminated by either a half word of Z's on the right or a full word of Z's.

The keys may extend over into the following block of the library if necessary. There is an 80-line (=160 keys) limit, however.

The key ZZZ000 produces no effect.

The lines referred to in the keys should be in ascending order of blocks.

6.3 The directory supplies the parameters to be inserted into the blanks.

The directory item which calls for the subroutine must carry in digits 4, 5, and 6 the number of directory lines bearing parameters which follow.

The lines of parameters are referred to in the keys relative to the call item which is considered to be line 00 in this scheme.

Whenever a subroutine to be processed under Mod I contains more than one block, a blank tape is required on servo 4 during compilation.

The compiler fills the blanks <u>first</u>, then compiles in the normal manner.

6.4 <u>Example</u>: A possible (though unlikely) library routine -

000	MF0	004	000	00~~1~~
001	10	000	30	000
002	30	000	30	000
003	60	000	-----	
004	~~BBB~~	~~BBB~~	~~BBB~~	~~BBB~~
005	001	W01	002	W02
006	003	L01	ZZZ	ZZZ

If the directory were to contain -

.

.

.

MF0	002	004	...
050	000	050	600
050	660	000	720

.

.

.

then, before being compiled, MF would become -

000	MF0	004	000	00~~1~~
001	15	000	35	600
002	35	660	30	720
003	65	000	-------	
004	~~BBB~~	~~BBB~~	~~BBB~~	~~BBB~~

6.5 A more compact treatment would be -

```
000   MF0 004  000 00Ø
001   10  000  30  000
002    -----   30  060
003   60  000  3ψ  120   (ψ is 4th zone zero.)
004   ØØØ ØØØ  ØØØ ØØØ
005   001 W01  002 R01
006   003 W01  ZZZ ZZZ
```

and -

```
MF0 001 004 ...
050 000 050 600
```

THE LIBRARIAN

1. The Librarian processes library tapes by inserting and deleting subroutines from the library and produces on the supervisory control typewriter a listing of the contents of the new library. It will also list without insertion or deletion.

2. Control for the Librarian is from the tape on servo 2. The following rules apply:

   A. If the first word of a block on tape 2 is a label line, tape 2 is copied into the new library in the alphabetically proper location. The number of blocks copied is indicated by the line count number in the label line.

   B. If the first word of a block on tape 2 is a word of minus signs a deletion is indicated. The following word on tape 2 identifies the routine which is to be deleted from the library by reproducing its label line.

   C. More deletions may be indicated in the same block on Servo 2 by preceding each label line with a word of minus signs.

   D. A word of zeros causes the next block on tape 2 to be read.

   E. A word of nines terminates tape 2 control.

   F. The routines inserted and deleted under tape 2 control must be dealt with in alphabetic order of label lines.

3. Servos.

   A. Inserting and deleting:

      Servo 2. Control tape

      6. Blank

      7. Old library

   B. Listing only:

      Servo 6. Library

4. Breakpoints. 1. Force transfer to change servos 6 and 7. The input tape is the old library the output tape is the blank to receive the new library. For listing only: input = output.

2. Force transfer to list only.

5. Supervisory control typewriter. normal

6. Ending procedure. When copying of the library is completed the old tape is rewound, the new one read backwards, then rewound, and the Univac stops on a 90 instruction. Press start bar to rewind tape 2; put skip in the static register to leave tape 2 positioned and call in the locator. The directory for a compilation may follow the Librarian control information on tape 2.

7. Abnormal typeouts.

"NO IGNORE [tape 2]": A word in label line position on tape 2 does not have a terminal ignore. Routine calls for a type-in to make correction.

"NO DELETE [tape 2]": Label lines on tape 2 and tape 7 do not match. Deletion request ignored by the Librarian.

8. Clear C to rerun.

SUBROUTINE CORRECTOR (LARRY I)

1. The subroutine corrector copies a library tape with manual corrections and compares the input and output tapes when copying is completed.

2. Servos. Any

3. Breakpoints.

   9. Force transfer to compare input and result tapes. Return typewriter carriage before forcing transfer.

4. S. C. Typewriter. Computer digit - 3 word (39 digits)

5. Type-ins.

   "INPUT-OUTPUT" : Input and output servo numbers

   "S-R AND WD" : 0000PQ000LLL where PQ is the routine label and LLL is the line of PQ to be corrected. Apply corrections in ascending order of blocks.

   : ZZZZZZZZZZZZ after last correction.

6. "END COPY 00BB". BB is the number of blocks in the library.

7. Clear C for rerun.

MIKE I

General Description:

Mike I is a routine designed to check a library tape, i.e., a tape of subroutines prepared for use in conjunction with the compiler. The routine checks for three things: (1) it checks to see if the subroutines appear on the tape in alphabetic order (if not, the typeout "tape not in alphabetic order" occurs); (2) it checks to see if the word of printer breakpoints (Ø) is where it should be in each subroutine--namely, on line corresponding to the number of lines in the subroutine as indicated in the label line (if not, the typeout "breakpoint misplaced" occurs); (3) it checks for proper use of the 9th digital position throughout each subroutine. IMPORTANT: All typeouts in category (3) do not necessarily indicate errors. They merely indicate where the programmer might have misused the 9th digital position. Many of these typeouts will be constants.

Operating Instructions:

(1) Set the typewriter on computer digit for two word typeouts (26 digits).

(2) When the routine types out AAAAAA OOONNN, it is calling for a control word of this form. For A type in the servo on which the subroutines to be processed are located. For N type in the number of subroutines to be processed (the two sentinal blocks at the beginning and end of the tape are not to be counted as subroutines.)

(3) Breakpoint 5 is used for skipping a subroutine without checking. If breakpoint 5 is depressed, the machine will stop after typing out the label line of each subroutine. Forcing transfer enables one to skip that particular subroutine and proceed to the next one. The normal path (no transfer) checks the subroutine.

(4) Rerun: Clear C and hit the start bar twice. The routine will rewind servo A and call for a new control word.

(5) When the routine types out "finished," it will automatically rewind servo A. Hitting the start bar twice will result in the routine calling for another control word.

INSTRUCTIONS FOR USE OF FLOATING DECIMAL - 1 (FD1)

1. FD1 is a floating decimal point arithmetic, single-precision, operating on real numbers. It occupies memory locations 880 to 999 (two blocks).

2. Representation of numbers

Numbers are represented in the form $X = x \cdot 10^{\alpha-50}$ where x lies in the range $0.1 \leq |x| < 1.0$ and α is the excess 50 exponent on 10.

a. Decomposed (two-word form):

$$\pm X = \begin{cases} \underline{0}\ x_1 \cdots x_{11}\ , & x_1 \neq 0 \\ 0\ \alpha_1\ \alpha_2\ 0 \cdots 0 \end{cases}$$

b. Composed (one-word form):

$$\pm X = \underline{0}\ \alpha_1\ \alpha_2\ x_1 \cdots x_9\ , \quad x_1 \neq 0$$

The character in the sign position determines the sign of x.

c. The representation of $X = 0$ is $x = 0$, $\alpha = 0$.

3. Overflow

FD1 employs a generalized overflow which transfers control to line c + 3 whenever overflow occurs on line c. The contents of register A are preserved.

IMPORTANT: Prior to use of FD1 it is necessary to transfer the contents of memory location 898 to memory location 000 in order to set the overflow constant.

4. Input to FD1

a. Decomposed:

(1) Place x and α in memory locations 982 and 983 respectively.

(2) Place y and β in memory locations 984 and 985 respectively, where $Y = y \cdot 10^{\beta-50}$.

b. Composed:

Place one-word arguments X and Y in register A and register X respectively.

Note: In both case a. and case b. the decomposed arguments X and Y will be in 982, 983 and 984, 985 respectively upon exit from FD1.

5. Output of FDI

a. Decomposed:

The results z and y will be found in memory locations 986 and 987 respectively, where

$$Z = z \cdot 10^{Y-50}.$$

b. Composed:

The result Z will be found in register A.

Note: In both case a. and case b. the decomposed result Z will be in memory locations 986, 987.

6. Operation paths

a. Decomposed exit:

(1) Addition X + Y = Z

(enter composed)	R	935	U	899
(enter decomposed)	R	935	U	900

(2) Multiplication X . Y = Z

(enter composed)	R	935	U	949
(enter decomposed)	R	935	U	950

(3) Division X/Y = Z

(enter composed)	R	935	U	938
(enter decomposed)	R	935	U	939

b. Composed exit:

After any of the operations in a., the order R 936 U 882 will put the composed result Z in register A. (See note under 5. Output of FDI.)

ALTERNATE PROCEDURE: If a composed exit path is desired for a succession of arithmetical operations, a short cut is to transfer the contents of memory location 882 to memory location 935. Then, use 936 as the address in any of the R orders under a. above in place of 935. This method for obtaining the composed exit option will remain valid provided an R 935 does not intervene to destroy the contents of memory location 935.

7. <u>Additional provisions in FT1</u>

a. Decomposition only:

(1) Put X into register A, Y into register M.

(2) R 890 U 884.

(3) x,α will be in 982, 983; y,β will be in 984, 985.

b. Composition only:

(1) Put z into 986, γ into 987.

(2) R 936 U 882.

(3) Result in register A.

c. Normalization:

A number is considered normalized when the most significant digit (x_1) is not zero. To normalize a floating point number:

(1) Decompose if necessary.

(2) Put z into 986, γ into register A.

(3) R 935 U 963.

(4) Decomposed result in 986, 987.

d. Conversion from fixed to floating point:

(1) Set γ equal to the scale factor plus 50.

(2) Put z into 986, γ into register A.

(3) R 935 U 963.

(4) Decomposed result in 986, 987.

Note: For a normal point number set γ = 50.

P-540527

AEC Computing Facility
New York University

Library of Subroutines

SUBROUTINE USE OF THE FLOATING DECIMAL ROUTINE (FD-1)

1. Subroutines in the library assume that the result of each floating decimal operation is used as input to the next operation. This is accomplished by executing V 986 W 982 in memory location 935. In order to insure the presence of this instruction in 935 FD-1 has been modified accordingly on the NYU Compiler System tape.

   Memory location 936 is henceforth the normal exit line for all decomposed floating operations. If memory location 935 is altered to contain other than V 986 W 982 it must be restored prior to using any of the library routines. The order B 989 C 935 will accomplish this.

2. The following working storage and constants are available in FD-1:

   Working Storage

   976
   977
   978
   979
   980
   981

   Constants

881	000000000000	
891	100000000000	
897	050000000000	
914	009000000000	
917	-10000000000	
918	010000000000	
919	S00911000000	(left hand address extractor,
929	001000000000	equivalent to 000111000000)
937	051000000000	
945	-000000000050	
955	099999999999	
958	000000000000	
962	000000000002	
969	000000000111	
990	011000000000	

Date Due

MAR 19 1971			
APR 26 1972			
	PRINTED	IN U. S. A.	

NYU
NYO-
6478

AUTHOR Goldfinger

TITLE New York University
compiler system.

NYU
NYO
6478

AUTHOR Goldfinger

TITLE New York University
compiler system.

DATE LOANED	BORROWER'S NAME	DATE RETURNED

NYU
NYO-
6478

Goldfinger

New York University
compiler system.